El modelo de Programación de Actor aplicado a Edge Computing utilizando Calvin

Nelson Rodríguez , María Murazzo, Tatiana Runco

Departamento de Informática, Facultad de Ciencias Exactas Físicas y Naturales,
Uniuversidad Nacional de San Juan, San Juan, Argentina

Resumen. Internet de las cosas (IoT) y todas las tecnologías asociadas han determinado que la computación distribuida alcance límites insospechables. A pesar de esta expansión, el desarrollo de aplicaciones en ecosistemas IoT no cumple con los requisitos específicos para estos sistemas. La pérdida de conectividad de los dispositivos, la necesidad de soportar la migración de código, la aparición de diversos errores y que los mismos no se propaguen, determinan que los desarrollos deben considerar estas dificultades. Teniendo en cuenta que el modelo de programación de actor presenta características que pueden ayudar a resolver estos problemas, se propuso utilizar este modelo para generar soluciones IoT, desarrollando aplicaciones con la entorno de aplicación Calvin. El presente trabajo analiza las ventajas de aplicar la programación de actores en ambientes IoT.

Keywords: Actor model programming, Edge Computing, Calvin

1 Introducción

Internet de las cosas (IoT) tiene un rico legado tecnológico y un brillante futuro: la conectividad ubicua ha creado un nuevo paradigma, y los sistemas cerrados, estáticos y limitados del pasado pronto serán obsoletos. Con la conexión de sensores de bajo costo, y la plataforma en el Cloud, ahora es posible realizar un seguimiento, analizar y responder a los datos operativos a gran escala [11].

Las aplicaciones de Internet de las cosas, llegan con un cambio de paradigma comparado con las aplicaciones de la Internet tradicional, debido a que ahora se generan enormes cantidades de datos en el borde de la red y fluye hacia la nube para ser procesados y potencialmente combinado con otros datos de diferentes fuentes [12].

Mientras que la primera generación de aplicaciones IoT a menudo seguían una arquitectura cliente-servidor tradicional, con clientes delgados y de baja complejidad, dejando que las tareas de alto costo computacional se realizaran en el Cloud, las aplicaciones recientes se caracterizan por clientes más ricos (teléfonos inteligentes, drones, automóviles) que utilizan datos de sensores locales como entrada para -posiblemente basado en aprendizaje automático- realizar el procesamiento de datos para respaldar la toma de decisiones a tiempo y local (por ejemplo para control

vehicular) o interacción del usuario, por ejemplo, para aplicaciones de realidad aumentada [7].

Se ha comenzado a comprobar que en la actualidad Cloud Computing está encontrando serias dificultades para satisfacer los requerimientos de IoT. Por ello, surgió como propuesta de solución llevar el almacenamiento, las funciones de red, gran parte del procesamiento hacia el borde de la red, lo que resultó en un nuevo modelo llamado edge computing [6].

En particular si se realiza en los dispositivos, sensores y actuadores se denomina mist computing, mientras que cuando se desarrolla en el extremo medio (fundamentalmente routers o switches de red) se denomina fog computing.

La programación para este tipo de sistemas se realiza utilizando lenguajes convencionales, los cuales presentan varias limitaciones para desarrollar aplicaciones para IoT. Los mismos no son de naturaleza asincrónica, no soportan aislación de errores y los mismos se pueden propagar (es común tener errores debido a descargas de baterías o falla de conexión por alcance, entre otras causas). Por lo tanto es conveniente utilizar otro tipo de lenguajes, herramientas o frameworks que permitan desarrollar aplicaciones eficientes para estos entornos.

El modelo de programación de actor presenta características que dan respuesta a esta problemática y la propuesta del presente trabajo es transmitir la experiencia en el desarrollo de aplicaciones para IoT basadas en este modelo.

El trabajo se encuentra organizado de la siguiente manera: se definen los conceptos más relevantes para el trabajo de IoT y del modelo de programación de actor, para posteriormente mostrar la experiencia en el desarrollo utilizando el entorno de aplicaciones Calvin y exponer las conclusiones y propuestas futuras pertinentes.

2 IoT

Internet de las cosas se puede definir como un conjunto de diferentes sistemas (por ejemplo, Smart Roads, Smart Buildings, Smart Grids) compuestos de componentes heterogéneos pero interactivos (por ejemplo, humanos, automóviles, teléfonos inteligentes, gateways, medidores inteligentes), tanto individual como colectivamente, que proporcionan servicios cibernéticos innovadores. Es decir, se puede describir como un ecosistema denso, a gran escala, abierto y dinámico de entidades y aplicaciones socio técnicas [22].

La idea básica de IoT es permitir una conexión e intercambio de datos autónomos y seguros entre dispositivos del mundo real y sus aplicaciones [10]. IoT vincula la vida real y las actividades físicas con el mundo virtual [16].

Hoy en día, IoT puede incluir productos industriales y comerciales, productos cotidianos como lavaplatos y termostatos, y redes locales de sensores para monitorear granjas y ciudades. En una solución de IoT, los objetos se pueden detectar y controlar a través de Internet, ya sea que se trate de dispositivos remotos, productos inteligentes o sensores que representan el estado de una ubicación física. Y la información puede estar disponible para aplicaciones, data warehouses y sistemas de negocios.

Sin embargo, a pesar de una década de investigación, el IoT aún se encuentra en una fase emergente: de hecho, está formado por unos pocos sistemas / dispositivos aislados de IoT que proporcionan servicios de computación convencionales diseñados principalmente para entornos estáticos con interacciones diseñadas a priori [5].

Los investigadores consideran que existe una tendencia clara y prominente hacia una IoT de mayor alcance, en la que posiblemente los impulsores clave serán los servicios ciberfísicos, altamente dinámicos y contextualizados.

Los bloques de construcción de la IoT son de baja potencia, integrados en dispositivos, donde las redes más grandes se construyen a partir de estos nodos pequeños y exhiben un flujo de trabajo altamente distribuido. Además de las restricciones del entorno de hardware, los desarrolladores tienen que tratar con un diseño de aplicación altamente distribuido [15].

Si bien las restricciones de hardware incluyen una baja potencia de procesamiento, paquetes de pequeño tamaño, una gran pérdida de paquetes y fallas temporales de conexión, los entornos distribuidos plantean desafíos con respecto a la escalabilidad y movilidad.

Por otro lado, los aspectos de seguridad deben considerarse muy seriamente, ya que estos dispositivos tienen acceso a una amplia gama de datos y la arquitectura de seguridad de este tipo de redes no es la más confiable. Cualquier persona que se encuentre cerca puede escuchar la transmisión inalámbrica y, por lo tanto, debe estar protegida.

Por ejemplo se podrían insertar con facilidad mensajes falsos, lo cual es una amenaza que puede ser contrarrestada si se implementa la autenticación de nodos de confianza. Un modelo de autenticación adecuado debe incluir la posibilidad de revocar la confianza de los nodos dañados o supuestamente atacados.

El uso generalizado de IoT crea nuevos desafíos, y su interacción sin precedentes con el mundo físico afecta la seguridad y la privacidad de las personas. El derecho a la privacidad debe estar protegido en el dispositivo, durante las comunicaciones y en el Cloud. Como regla general, la información debe transferirse y accederse según sea necesario y se debe cambiar, y los eventos como la creación de datos, el acceso a datos y los comandos de control deben registrarse de forma verificable [9].

Las aplicaciones pueden verse como una combinación de microservicios que se utilizan para crear un servicio. Estas aplicaciones pueden ubicarse estáticamente o migrarse dinámicamente al entorno que sea óptimo para su realización. La seguridad de las aplicaciones será el resultado del propio código de la aplicación y la plataforma que está utilizando. En los casos en que las aplicaciones pueden migrar, es importante que la migración entre plataformas se realice de forma segura [19]. En los sistemas de Cloud, las aplicaciones se pueden colocar de forma segura en plataformas confiables mediante el uso de información certificada que proviene de la confianza en la infraestructura del Cloud.

El futuro de IoT tiene el potencial ilimitado. Los avances en Internet industrial se acelerarán a través de una mayor agilidad de la red, la inteligencia artificial integrada y la capacidad de desplegar, automatizar, organizar y asegurar diversos casos de uso a nivel de hiperescala. El potencial no es solo habilitar miles de millones de dispositivos simultáneamente, sino también aprovechar los enormes volúmenes de datos procesables que pueden automatizar diversos procesos de negocios. A medida que las redes y las plataformas de IoT evolucionan para superar estos desafíos, a

través de una mayor capacidad e inteligencia artificial, los proveedores de servicios se adentran aún más en los mercados de TI y de escala web, lo que abre nuevos flujos de ingresos.

3 El Modelo de Programación de Actor

El modelo Actor es una teoría matemática de la computación que trata a los "Actores" como primitivas universales de la computación digital concurrente [18]. El modelo ha sido utilizado tanto como un framework para una comprensión teórica de la concurrencia, y como la base teórica para varias implementaciones prácticas de sistemas concurrentes.

A diferencia de los modelos de computación anteriores, el Modelo de actor se inspiró en las leyes físicas. También estuvo influenciado por los lenguajes de programación Lisp, Simula-67 y Smalltalk72, así como por fundamentos de redes de Petri, sistemas de capacidad y packet switching. El advenimiento de la concurrencia masiva a través de la computación en la nube y las arquitecturas de computadora multi núcleo ha estimulado el interés en el Modelo de actor.

Dicho modelo de programación fue definido por Carl Hewitt en 1973, y presenta ventajas considerables para el desarrollo de aplicaciones para IoT [18].

En IoT los dispositivos pueden dejar de tener conexión momentáneamente por diversas razones, por ejemplo la intensidad de la señal, razones climáticas, descarga de baterías (baterías solares) y posiblemente tiempo posterior vuelvan a conectarse, debido a la recarga de la batería solar o al alcance de la señal, si es un dispositivo móvil. Esta conexión y desconexión produce fallos que pueden propagar otros fallos en el resto de la red. En el modelo de programación de actor, cada actor se comunica de forma asincrónica con otro lo cual facilita la comunicación cuando hay conexión - desconexión de los dispositivos. Por otro lado, los actores presentan independencia en la ejecución unos de otros, lo cual en caso de falla no la propagaría al resto de la red. Además son muy livianos, por ejemplo en algunos toolkits como Akka ocupan solamente 600 Bytes, lo cual lo hace muy adecuado para entornos donde los dispositivos tienen poca capacidad de procesamiento y memoria

El modelo de programación de actor permite la concurrencia, está basado en memoria distribuida y puede ser aplicado al edge [14].

El modelo de actor [1] ofrece varias ventajas sobre el modelo de threads. Los actores modelan el mundo real de manera intuitiva, y escribir aplicaciones confiables utilizando actores es simple. La comunicación asíncrona entre actores conduce a una mejor utilización del tiempo de CPU, ya que los ciclos de inactividad no se desperdician mientras se espera una respuesta.

El modelo de programación de actor presenta mínimas diferencias con la programación reactiva.

Los objetos reactivos tienen las siguientes ventajas sobre los actores:

Los mensajes a métodos no definidos son simplemente puestos en cola.

El modelo de actor carece de mensajes síncronos.

La entrega asíncrona de mensajes no preserva el orden.

Sin embargo, la sincronización puede ser emulada por pares de mensajes asíncronos, y la mayoría de las implementaciones modernas presentan mejoras en estos los tres puntos

4 Sistemas de Actor Disponibles

Existen diversos lenguajes de programación, librerías y otras herramientas de software que permiten generar aplicaciones basadas en el modelo de programación de actor. Se van a describir las más emblemáticas.

Erlang [8] fue uno de los primeros lenguajes que soportan el modelo. Implementa actores a nivel de lenguaje y los llama procesos. Los procesos son ligeros, ya que se implementan a nivel de la máquina virtual Erlang, y no implican hilos o procesos de un sistema operativo. Cada actor se le asigna su propia memoria dinámica, y la recolección de basura se realiza para cada actor independientemente. Los mensajes enviados se copian entre montones, lo que impide compartir ellos por dos o más actores. La programación de actores es un deber de la máquina virtual Erlang, y no se necesita ningún mecanismo de sincronización, por ejemplo, semáforos. El Sistema de actores garantiza la escalabilidad (un actor necesita solo 300 bytes de memoria) y una alta confiabilidad de los programas escritos.

Es un lenguaje de programación funcional de alto nivel, que se ubica dentro del paradigma de Programación Declarativa, diseñado para escribir aplicaciones concurrentes y distribuidas de funcionamiento ininterrumpido.

Erlang usa procesos concurrentes para estructurar la aplicación. Estos procesos no comparten memoria y se comunican de forma asincrónica mediante el paso de mensajes. Los procesos Erlang son muy ligeros y pertenecen al propio lenguaje, no al sistema operativo. También posee un mecanismo para cambiar el código fuente de un programa o aplicación en ejecución (cambio en caliente de código), sin tener que detener el programa. Este mecanismo facilita la implementación de sistemas indetenibles [3,4].

Los inicios de este lenguaje toman lugar en los años 80 en los laboratorios de Ciencias de Computación de la Compañía de telefonía sueca Ericsson.

Akka [2] es un popular framework de actores al tope de la JVM escrito en Scala. Desde la versión 2.10.0, Scala usa Akka como la biblioteca de actores predeterminada [17].

El paso del mensaje es hecho en memoria compartida, si los actores se ejecutan dentro de una máquina virtual Java. Si los actores actúan en diferentes máquinas virtuales, los mensajes se serializan antes de enviarlos. La recolección de basura se aplica a todos los actores de una instancia de JVM determinada. Los actores están programados por la biblioteca Akka. No impone mensajes inmutables y evita el estado global, aunque los recomienda [24].

Al igual que Erlang, Akka adopta el modelo "Let it crash" para la resiliencia.

El cluster Akka proporciona un servicio de membresía de clúster que se basa en un protocolo Gossip, similar a Dynamo, el almacén de valores y claves distribuido de Amazon. El clúster Akka usa los latidos del corazón en un intervalo de 1 s y el Detector de fallas de acumulación de phi para calcular la probabilidad de que un nodo

sea inalcanzable. Los programadores pueden modificar la detección de fallas estableciendo un valor de umbral. Lightbend se ha ejecutado con éxito un clúster Akka con 2400 nodos en Google Compute Engine [23].

Calvin [20] está diseñado principalmente para simplificar el desarrollo de aplicaciones de Internet de las cosas y está disponible en Github https://github.com/EricssonResearch/calvin-base .

Fue creado por Ericsson basado en experiencia previa de Erlang, es un entorno de aplicación que permite a las cosas hablar con las cosas. Incluye un marco de desarrollo para programadores de aplicaciones y un entorno de ejecución para el manejo de la aplicación en ejecución.

Calvin se basa en la idea fundamental de que el desarrollo de aplicaciones debe ser simple y divertido. No debe haber barreras innecesarias entre una idea y su implementación. Un desarrollador de aplicaciones no debería tener que preocuparse por los protocolos de comunicación o detalles de hardware [21].

Hay que tener en cuenta, además que estos lenguajes basados en actores también son eficientes para servidores, Un resumen de las recomendaciones basadas en este pequeño conjunto de puntos de referencia es el siguiente: Para un servidor donde minimizar la latencia de mensajes es crucial Erlang es una buena opción. Por otro lado Akka reduce significativamente la latencia de comunicación en Scala y es capaz de mantener el mayor número de procesos inactivos, mientras que Erlang tiene el mejor desempeño cuando los procesos son de corta duración y el objetivo es garantizar un tiempo de generación mínimo, por ejemplo [23].

Se encuentran disponibles otros productos de menor relevancia o uso como: Pony, Concurrent ML, Anemone, Orleans, Orbit, SF Reliable Actors, entre otros.

5 Ventajas de Calvin en ecosistemas IoT

Calvin presenta diversas ventajas para el desarrollo y mantenimiento de aplicaciones IoT:

Tiene la capacidad de ejecutar partes del código de la aplicación donde sea más beneficioso por ejemplo, en un nodo central al analizar datos agregados de muchas fuentes o cerca de hardware específico para lograr baja latencia.

Calvin se basa en el modelo de actor bien establecido, utilizando una metodología a menudo denominada programación de flujo de datos.

Es escalable, la funcionalidad básica es bastante simple y funciona en dispositivos IoT pequeños, aun así utiliza la energía de cálculo completa disponible en el cloud.

El modelo de programación de Calvin no hace distinción entre el cloud y el dispositivo, smartphone y sensor, cliente y servidor, todos están representados por actores y comparten el mismo paradigma. Al escribir aplicaciones de Calvin, no hay necesidad de preocuparse por exactamente dónde desplegarlo mientras se cumplan los requisitos de hardware necesarios. El desarrollo y el despliegue son preocupaciones independientes.

Mover una aplicación a otro conjunto de nodos de cálculo no requiere cambiar el código. Por el contrario, los cambios en la aplicación no impiden la implementación en el mismo conjunto de nodos de cálculo.

Los actores de Calvin pueden incluso hacer la migración en vivo de un runtime (tiempo de ejecución) a otro. El sistema lo logra serializando el estado interno del actor en movimiento, enviándolo al tiempo de ejecución al destino en el que se des-serializa y carga una copia del actor en movimiento. Esta propiedad hace posible escribir aplicaciones que se adaptan a las condiciones cambiantes. Por lo tanto, un runtime en un dispositivo que necesita entrar en modo de suspensión puede mover a los actores fuera de sí mismo sin interrumpir la aplicación de la que son parte de antes de entrar en modo suspensión.

Según especialistas de Ericsson, Calvin está pensado para todos los desarrolladores de Internet de las Cosas, debido al modelo de programación simple, la potente comunicación y a los mecanismos de despliegue que están incorporados en el sistema. Las complejidades están ocultas, el desarrollo se vuelve más eficiente, el resultado es más robusto y hay menos errores en el código.

Un modelo de programación específicamente diseñado para un entorno de ejecución distribuida es fundamental para hacer que la Internet de las cosas realmente vuele. Calvin es una tecnología muy prometedora que satisface estas necesidades.

6 Aplicaciones desarrolladas

Se desarrollaron varias aplicaciones para evaluar las ventajas de aplicar este modelo en entornos de IoT. En todos los casos se trabajó sobre la placa computadora “Raspberry Pi” conectada a sensores que recopilan los datos necesarios para la aplicación práctica a desarrollar. Esta plataforma cuenta con capacidad de procesamiento y todas las E/S necesarias, conexión Wi-Fi, Bluetooth y otras funcionalidades, además se le instaló el sistema operativo Raspbian.

La más relevante y de la cual se obtuvieron muchas conclusiones es una aplicación diseñada para mejorar la producción avícola en la incubación de huevos de aves, específicamente huevos de gallina. Debido al riesgo de las vidas de las aves en el momento del desarrollo embrionario, se presentó una propuesta de diseño para un sistema de incubación que busca cubrir aspectos principalmente de funcionalidad acordes al desarrollo embrionario del huevo de gallina, lo cual ayuda a disminuir los costos de producción antes mencionados.

Una incubadora avícola tiene como principal objetivo mantener las condiciones adecuadas para el nacimiento del ave que se pretenda incubar, basándose en un análisis fenomenológico que ocurre dentro del periodo de incubación. Para ello debe realizarse antes una revisión de los aspectos más relevantes que determinan la incubabilidad de los huevos. Como son los aspectos de temperatura y humedad relativa de incubación que competen a los fines prácticos del software a desarrollar.

Dicha aplicación se desarrolló en Calvin y permitió comprobar las ventajas del modelo para entornos de IoT como son:

- Evita el bloqueo de la programación orientada a objetos con el paso de mensajes asíncronos pero sigue conservando el principio de encapsulamiento.

- Como ya no existe una pila de llamadas compartidas entre actores que se envían mensajes entre sí, las situaciones de error se manejan por paso de mensajes y el servicio de supervisión.
- La migración de Actores se puede lograr en tiempo de ejecución entre diferentes aplicaciones.
- Facilita considerablemente el trabajo del programador a la hora de desarrollar aplicaciones gracias a la simplicidad del mismo lenguaje y gracias a que el mismo lenguaje se encarga de todo el proceso de comunicación.

7 Conclusiones

Calvin se pudo instalar sin dificultad en cada dispositivo que fue propuesto, facilita la flexibilidad y adaptabilidad al fusionar el Edge en IoT y el Cloud. La programación se distribuye sin complicaciones, combina la simplicidad y la reutilización de la funcionalidad. Por otro lado sobre Rapberry y Raspbian como sistema operativo funcionó adecuadamente y con muy baja latencia.

Implementa todos los principios y características del Modelo de Programación de Actor mencionados anteriormente, permitiendo la creación de aplicaciones que ofrecen ventajas significativas sobre el entorno de IoT: migración en caliente de Actores, sistemas más escalables, distribuidos y simples de desarrollar para los programadores (abstrayéndolos de protocolos de comunicación o detalles de hardware), las cuales no soportan los modelos de programación convencionales.

La única desventaja es que Calvin es muy sensible en sus archivos de configuración. Es decir, cada cambio que se desee realizar difiriendo de los que ya se encuentran por defecto, implica la modificación de los archivos de configuración del mismo y debiéndose considerar que tal archivo de configuración se encuentre en la ruta donde se ejecute el tiempo de ejecución que haga uso de la configuración deseada. Por ejemplo: para lograr el correcto funcionamiento de los sensores, actores y de la Raspberry Pi.

El desarrollo de aplicaciones con el modelo de actor debe ser planificado adecuadamente desde etapas iniciales, dado que si bien se dispone de herramientas de software para la construcción de las mismas que soportan esta forma de programación, no es suficiente, es decir se puede programar en Akka o Calvin y no necesariamente el resultado tiene toda la funcionalidad y las ventajas de un desarrollo basado en el modelo de actor.

Debido a las características de Calvin, se puede construir un Cloud distribuido, con importante funcionalidad en el edge, mediante esta arquitectura se puede ayudar a lograr una baja latencia mediante el uso de nodos de procesamiento paralelos.

Existen otros frameworks desarrollados para IoT, sin embargo, Calvin simplifica el trabajo para el desarrollador de la aplicación. El tiempo de ejecución oculta gran parte del trabajo complejo y el desarrollador de la aplicación no necesita preocuparse por cosas como la migración de actores o la transmisión de mensajes.

La primera generación de IoT se trató de la transición desde M2M, sin embargo actualmente IoT es una amalgama de diferentes disciplinas, como los sistemas cibernéticos, redes de sensores inalámbricos, M2M y Big Data / Inteligencia Artificial, en un contexto singular de una Internet del mundo real. A lo largo de los años, el ecosistema ha crecido de manera exponencial, y hoy, naturalmente, incluye no solo a las empresas en el dominio de TI, sino también a OT - Tecnología operacional: el nombre utilizado para las tecnologías asociadas con el monitoreo y control de dispositivos físicos y procesos en entornos industriales y de fabricación. Cualquier jugador importante de TI u OT en el mercado ahora está en desventaja si no tiene productos o servicios de soporte basados en IoT.

Una estrategia clave en la arquitectura futura, es el papel que jugará el extremo (Edge), dado que fog computing cada vez adquiere mayor relevancia y seguramente será parte del soporte para IoT. El rol del extremo es muy importante para reducir la latencia de ir al Cloud, que en casos de soluciones dependiente del tiempo puede resultar muy ventajosa, y en este caso se puede visualizar las ventajas de aplicar la programación de actor en estos contextos.

8 Trabajos futuros

Internet de las cosas (IoT) se está expandiendo rápidamente y según algunas estimaciones se espera que comprenda 18 mil millones de dispositivos conectados para 2022.

Las suposiciones de confianza y de buenas intenciones por parte de los usuarios, que formaron parte del desarrollo temprano de Internet, no se pueden aplicar en las primeras etapas del desarrollo de IoT. La privacidad y las preocupaciones de seguridad son cada vez mayores, especialmente dada la creciente importancia de IoT en los contextos corporativo, gubernamental y de infraestructura crítica. Del mismo modo, la mercantilización de los componentes de IoT incorporados en diversos rangos de productos y desplegados tanto en casos de uso administrados como no administrados presenta importantes desafíos de seguridad y crea potencial para nuevos tipos de ataques. La cooperación proactiva de todas las partes interesadas clave será necesaria para obtener los considerables beneficios económicos del IoT, al tiempo que protege la seguridad, la seguridad y la privacidad.

Se espera realizar estudios tendientes a valorar la adopción del modelo de actor para proveer software más seguro y administrar de forma más eficiente la seguridad provista.

Referencias

1. Agha G. A. :ACTORS - a model of concurrent computation in distributed systems. MIT Press series in artificial intelligence. Cambridge (1990).
2. Akka : Build highly concurrent, distributed, and resilient message-driven applications on the jvm. https://github.com/akka/akka, 2018.

3. Armstrong J. : Erlang- software for a concurrent world. In: Ernst E (ed.) ECOOP 2007. Object-Oriented Programming, 21st European Conference, Berlin, Germany, Proceedings, vol. 4609, p 1 (2007).
4. Armstrong J. : Erlang. Commun ACM Vol. 53 N. 9, pp. :68–75 (2010).
5. Biron J., Follett J.: Foundational Elements of an IoT Solution The Edge, The Cloud, and Application Development, O'Reilly Media, Inc. (2016).
6. Bonomi F. et al.: Fog Computing and Its Role in the Internet of Things. In: Proc. 1st Edition MCC Workshop Mobile Cloud Computing (MCC12), pp.13–15. (2012).
7. Chen K., J. Furst, Kolb J., Kim H., Jin X., Culler D., Katz R.: Snaplink: Fast and accurate vision-based appliance control in large commercial buildings. In: Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies, vol. 1, no. 4, p. 129 (2018).
8. Erlang: Sitio oficial: https://www.erlang.org/
9. Ericsson: IoT security - protecting the networked society. Jun 1, 2017 | White paper, https://www.ericsson.com/en/white-papers/iot-security-protecting-the-networked-society
10. Fan T., Chen Y.: A Scheme of Data Management in the Internet of Things. In: 2nd IEEE International Conference on Network Infrastructure and Digital Content. (2010).
11. Fortino G., Russo W., Savaglio C., Viroli M., Zhou M.: Opportunistic cyberphysical services: A novel paradigm for the future Internet of Things, in: 2018 IEEE 4th World Forum on Internet of Things (WF-IoT), 2018, pp. 488–492, http://dx.doi.org/10.1109/WF-IoT.2018.8355174. (2018).
12. Furst J., Fadel Argerich M., Chen K., Kovacs E.: Towards Adaptive Actors for Scalable IoT Applications at the Edge", Open Journal of Internet of Things (OJIOT), Volume 4, Issue 1, (2018)
13. Hewitt C. : Actor Model of Computation: Scalable Robust Information Systems, Cornell University, https://arxiv.org/abs/1008.1459 (2015).
14. Hewitt C., Bishop P. and Steiger R. : A Universal Modular ACTOR Formalism for Artificial Intelligence. In: Proceedings of the 3rd international joint conference on Artificial (1973).
15. Hiesgen R., Charousset D., Schmidt T. C. : Embedded Actors – Towards Distributed Programming in the IoT. 978-1-4799-6165-8/14/ IEEE. (2014).
16. Huang Y., Li G.: A Semantic Analysis for Internet of Things. In: International Conference on Intelligent Computation Technology and Automation (ICICTA), (2010).
17. Jovanovic V., Haller P. : The scala actors migration guide.
http://docs.scala-lang.org/overviews/core/actors-migration-guide.html.
18. Patil S., Sruthi N. S. , Narkhede, B. E. , Pendam D. V. : A 20/20 vision of Internet of things. IOSR Journal of Business and Management. 18. 76-80. 10.9790/487X-1808027680 (2016).
19. Persson et. al. : Calvin – Merging Cloud and IoT, http://www.sciencedirect.com/science/article/pii/S1877050915008595
20. Perssson J.: Open Source release of IoT app environment Calvin. https://www.ericsson.com/research-blog/open-source-calvin/ (2015).
21. Persson P., Angelsmark, O.: Calvin–merging Cloud and IoT. In: Procedia Computer Science, 52, pp. 210-217 (2015).
22. Savaglio C., Fortino G., Zhou M.: Towards interoperable, cognitive and autonomic IoT systems: an agent-based approach. In: Internet of Things (WF-IoT), 2016 IEEE 3rd World Forum on, IEEE, pp. 58–63. (2016).
23. Valkov I., Chechina N., Trinder P.: Comparing Languages for Engineering Server Software: Erlang, Go, and Scala with Akka, SAC 2018: Symposium on Applied Computing. Pau, France (2018).
24. Wyatt D. : Akka concurrency. Artima Incorporation, New York (2013).